StringBuffer class in Java

StringBuffer is a class in Java that represents a mutable sequence of characters. It provides an alternative to the immutable String class, allowing you to modify the contents of a string without creating a new object every time.

Here are some important features and methods of the StringBuffer class:

1. StringBuffer objects are mutable, meaning that you can change the contents of the buffer without creating a new object.
2. The initial capacity of a StringBuffer can be specified when it is created, or it can be set later with the ensureCapacity() method.
3. The append() method is used to add characters, strings, or other objects to the end of the buffer.
4. The insert() method is used to insert characters, strings, or other objects at a specified position in the buffer.
5. The delete() method is used to remove characters from the buffer.
6. The reverse() method is used to reverse the order of the characters in the buffer.

Here is an example of using StringBuffer to concatenate strings:

Java

```
public class StringBufferExample {
    public static void main(String[] args)
    {
        StringBuffer sb = new StringBuffer();
        sb.append("Hello");
        sb.append(" ");
        sb.append("world");
        String message = sb.toString();
        System.out.println(message);
    }
}
```

Output

```
Hello world
```

There are several advantages of using StringBuffer over regular String objects in Java:

1. Mutable: StringBuffer objects are mutable, which means that you can modify the contents of the object after it has been created. In contrast, String objects are immutable, which means that you cannot change the contents of a String once it has been created.
2. Efficient: Because StringBuffer objects are mutable, they are more efficient than creating new String objects each time you need to modify a string. This is especially true if you need to modify a string multiple times, as each modification to a String object creates a new object and discards the old one.
3. Thread-safe: StringBuffer objects are thread-safe, which means multiple threads cannot access it simultaneously. In contrast, String objects are not thread-safe, which means that you need to use synchronization if you want to access a String object from multiple threads.

Overall, if you need to perform multiple modifications to a string, or if you need to access a string from multiple threads, using StringBuffer can be more efficient and safer than using regular String objects.

StringBuffer is a peer class of **String** that provides much of the functionality of strings. Th string represents fixed-length, immutable character sequences while StringBuffer represent growable and writable character sequences. **StringBuffer** may have characters and substring inserted in the middle or appended to the end. It will automatically grow to make room for suc additions and often has more characters preallocated than are actually needed, to allow room for growth.

StringBuffer class is used to create mutable (modifiable) string. The StringBuffer class in java is the same as String class except it is mutable i.e. it can be changed.

Important Constructors of StringBuffer class

- StringBuffer(): creates an empty string buffer with an initial capacity of 16.
- StringBuffer(String str): creates a string buffer with the specified string.
- StringBuffer(int capacity): creates an empty string buffer with the specified capacity as length.

1) append() method

The append() method concatenates the given argument with this string.

Example:

Java

```
import java.io.*;

class A {
    public static void main(String args[])
    {
        StringBuffer sb = new StringBuffer("Hello ");
        sb.append("Java"); // now original string is changed
        System.out.println(sb);
    }
}
```

Output

```
Hello Java
```

2) insert() method

The insert() method inserts the given string with this string at the given position.

Example:

Java

```
import java.io.*;

class A {
    public static void main(String args[])
    {
        StringBuffer sb = new StringBuffer("Hello ");
        sb.insert(1, "Java");
        // Now original string is changed
        System.out.println(sb);
    }
}
```

Output

```
HJavaello
```

3) replace() method

The replace() method replaces the given string from the specified beginIndex and endIndex-1.

Example:

Java

```
import java.io.*;

class A {
    public static void main(String args[])
    {
        StringBuffer sb = new StringBuffer("Hello");
        sb.replace(1, 3, "Java");
        System.out.println(sb);
    }
}
```

Output

```
HJavalo
```

4) delete() method

The delete() method of the StringBuffer class deletes the string from the specified beginIndex to endIndex-1.

Example:

Java

```
import java.io.*;

class A {
    public static void main(String args[])
    {
        StringBuffer sb = new StringBuffer("Hello");
        sb.delete(1, 3);
        System.out.println(sb);
    }
}
```

Output

```
Hlo
```

5) reverse() method

The reverse() method of the StringBuilder class reverses the current string.

Example:

Java

```
import java.io.* ;

class A {
    public static void main(String args[])
    {
        StringBuffer sb = new StringBuffer("Hello");
        sb.reverse();
        System.out.println(sb);
    }
}
```

Output

```
olleH
```

6) capacity() method

- The capacity() method of StringBuffer class returns the current capacity of the buffer. The default capacity of the buffer is 16. If the number of characters increases from its current capacity, it increases the capacity by (oldcapacity*2)+2.
- For instance, if your current capacity is 16, it will be (16*2)+2=34.

Example:

Java

```
import java.io.*;

class A {
    public static void main(String args[])
    {
        StringBuffer sb = new StringBuffer();
        System.out.println(sb.capacity()); // default 16
        sb.append("Hello");
        System.out.println(sb.capacity()); // now 16
        sb.append("java is my favourite language");
        System.out.println(sb.capacity());
        // Now (16*2)+2=34     i.e (oldcapacity*2)+2
    }
}
```

Output

```
16
16
34
```

Some Interesting Facts about the StringBuffer class

Do keep the following points in the back of your mind:

- All Implemented Interfaces of StringBuffer class: Serializable, Appendable, CharSequence.
- public final class StringBuffer extends Object implements Serializable, CharSequence, Appendable.
- String buffers are safe for use by multiple threads. The methods can be synchronized wherever necessary so that all the operations on any particular instance behave as if they occur in some serial order.
- Whenever an operation occurs involving a source sequence (such as appending or inserting from a source sequence) this class synchronizes only on the string buffer performing the operation, not on the source.
- It inherits some of the methods from the Object class which such as *clone(), equals(), finalize(), getClass(), hashCode(),* notifies(), *notifyAll().*

> ***Remember:*** *StringBuilder, J2SE 5 adds a new string class to Java's already powerful string handling capabilities. This new class is called StringBuilder. It is identical to StringBuffer except for one important difference: it is not synchronized, which means that it is not thread-safe. The advantage of StringBuilder is faster performance. However, in cases in which you are using multithreading, you must use StringBuffer rather than StringBuilder.*

Constructors of StringBuffer class

1. StringBuffer(): It reserves room for 16 characters without reallocation

```
StringBuffer s = new StringBuffer();
```

2. StringBuffer(int size): It accepts an integer argument that explicitly sets the size of the buffer.

```
StringBuffer s = new StringBuffer(20);
```

3. StringBuffer(String str): It accepts a string argument that sets the initial contents of the StringBuffer object and reserves room for 16 more characters without reallocation.

```
StringBuffer s = new StringBuffer("GeeksforGeeks");
```

Methods of StringBuffer class

Methods	Action Performed
append()	Used to add text at the end of the existing text.
length()	The length of a StringBuffer can be found by the length( ) method
capacity()	the total allocated capacity can be found by the capacity( ) method
charAt()	This method returns the char value in this sequence at the specified index.
delete()	Deletes a sequence of characters from the invoking object
deleteCharAt()	Deletes the character at the index specified by *loc*
ensureCapacity()	Ensures capacity is at least equals to the given minimum.
insert()	Inserts text at the specified index position
length()	Returns length of the string
reverse()	Reverse the characters within a StringBuffer object
replace()	Replace one set of characters with another set inside a StringBuffer object

> ***Note:*** *Besides that, all the methods that are used in the String class can also be used. These auxiliary methods are as follows:*

ensureCapacity(): It is used to increase the capacity of a StringBuffer object. The new capacity will be set to either the value we specify or twice the current capacity plus two (i.e. capacity+2), whichever is larger. Here, capacity specifies the size of the buffer.

Syntax:

```
void ensureCapacity(int capacity)
```

- **appendCodePoint(int codePoint):** This method appends the string representation of the codePoint argument to this sequence.

Syntax:

```
public StringBuffer appendCodePoint(int codePoint)
```

charAt(int index): This method returns the char value in this sequence at the specified index.

Syntax:

```
public char charAt(int index)
```

IntStream chars(): This method returns a stream of int zero-extending the char values from this sequence.

Syntax:

```
public IntStream chars()
```

int codePointAt(int index): This method returns the character (Unicode code point) at the specified index.

Syntax:

```
public int codePointAt(int index)
```

int codePointBefore(int index): This method returns the character (Unicode code point) before the specified index.

Syntax:

```
public int codePointBefore(int index)
```

int codePointCount(int beginIndex, int endIndex): This method returns the number of Unicode code points in the specified text range of this sequence.

Syntax:

```
public int codePointCount(int beginIndex, int endIndex)
```

IntStream codePoints(): This method returns a stream of code point values from this sequence.

Syntax:

```
public IntStream codePoints()
```

void getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin): In this method, the characters are copied from this sequence into the destination character array dst.

Syntax:

```
public void getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)
```

int indexOf(String str): This method returns the index within this string of the first occurrence of the specified substring.

Syntax:

```
public int indexOf(String str)
public int indexOf(String str, int fromIndex)
```

int lastIndexOf(String str): This method returns the index within this string of the last occurrence of the specified substring.

Syntax:

```
public int lastIndexOf(String str)
public int lastIndexOf(String str, int fromIndex)
```

int offsetByCodePoints(int index, int codePointOffset): This method returns the index within this sequence that is offset from the given index by codePointOffset code points.

Syntax:

```
public int offsetByCodePoints(int index, int codePointOffset)
```

void setCharAt(int index, char ch): In this method, the character at the specified index is set to ch.

Syntax:

```
public void setCharAt(int index, char ch)
```

void setLength(int newLength): This method sets the length of the character sequence.

Syntax:

```
public void setLength(int newLength)
```

CharSequence subSequence(int start, int end): This method returns a new character sequence that is a subsequence of this sequence.

Syntax:

```
public CharSequence subSequence(int start, int end)
```

String substring(int start): This method returns a new String that contains a subsequence of characters currently contained in this character sequence.

Syntax:

```
public String substring(int start)
public String substring(int start,int end)
```

String toString(): This method returns a string representing the data in this sequence.

Syntax:

```
public String toString()
```

void trimToSize(): This method attempts to reduce storage used for the character sequence.

Syntax:

```
public void trimToSize()
```

> *Above we only have discussed the most widely used methods and do keep a tight bound around them as they are widely used in programming geeks.*

Implementation:

Example 1: *length() and capacity()* Methods

Java

```
// Java Program to Illustrate StringBuffer class
// via length() and capacity() methods

// Importing I/O classes
import java.io.*;

// Main class
class GFG {

    // main driver method
    public static void main(String[] args)
    {

        // Creating and storing string by creating object of
        // StringBuffer
        StringBuffer s = new StringBuffer("GeeksforGeeks");
```

```
        // Getting the length of the string
        int p = s.length();

        // Getting the capacity of the string
        int q = s.capacity();

        // Printing the length and capacity of
        // above generated input string on console
        System.out.println("Length of string GeeksforGeeks="
                           + p);
        System.out.println(
            "Capacity of string GeeksforGeeks=" + q);
    }
}
```

Output

```
Length of string GeeksforGeeks=13
Capacity of string GeeksforGeeks=29
```

Example 2: append()

It is used to add text at the end of the existing text.

Here are a few of its forms:

```
StringBuffer append(String str)
StringBuffer append(int num)
```

Java

```
// Java Program to Illustrate StringBuffer class
// via append() method

// Importing required classes
import java.io.*;

// Main class
class GFG {

    // Main driver method
    public static void main(String[] args)
    {
        // Creating an object of StringBuffer class and
        // passing random string
        StringBuffer s = new StringBuffer("Geeksfor");

        // Usage of append() method
        s.append("Geeks");

        // Returns GeeksforGeeks
        System.out.println(s);

        s.append(1);
        // Returns GeeksforGeeks1
```

```
        System.out.println(s);
    }
}
```

Output

```
GeeksforGeeks
GeeksforGeeks1
```

Example 3: insert()

It is used to insert text at the specified index position.

Syntax: These are a few of its as follows:

```
StringBuffer insert(int index, String str)
StringBuffer insert(int index, char ch)
```

Here, the *index* specifies the index at which point the string will be inserted into the invoking **StringBuffer** object.

Java

```
// Java Program to Illustrate StringBuffer class
// via insert() method

// Importing required I/O classes
import java.io.*;

// Main class
class GFG {

    // Main driver method
    public static void main(String[] args)
    {
        // Creating an object of StringBuffer class
        StringBuffer s = new StringBuffer("GeeksGeeks");

        // Inserting element and position as an arguments
        s.insert(5, "for");
        // Returns GeeksforGeeks
        System.out.println(s);

        s.insert(0, 5);
        // Returns 5GeeksforGeeks
        System.out.println(s);

        s.insert(3, true);
        // Returns 5GetrueeksforGeeks
        System.out.println(s);

        s.insert(5, 41.35d);
        // Returns 5Getr41.35ueeksforGeeks
        System.out.println(s);
```

```
        s.insert(8, 41.35f);
        // Returns 5Getr41.41.3535ueeksforGeeks
        System.out.println(s);

        // Declaring and initializing character array
        char geeks_arr[] = { 'p', 'a', 'w', 'a', 'n' };

        // Inserting character array at offset 9
        s.insert(2, geeks_arr);
        // Returns 5Gpawanetr41.41.3535ueeksforGeeks
        System.out.println(s);
    }
}
```

Output

```
GeeksforGeeks
5GeeksforGeeks
5GetrueeksforGeeks
5Getr41.35ueeksforGeeks
5Getr41.41.3535ueeksforGeeks
5Gpawanetr41.41.3535ueeksforGeeks
```

Example 4: reverse( )

It can reverse the characters within a StringBuffer object using **reverse( ).** This method returns the reversed object on which it was called.

Java

```
// Java Program to Illustrate StringBuffer class
// via reverse() method

// Importing I/O classes
import java.io.*;

// Main class
class GFG {

    // Main driver method
    public static void main(String[] args)
    {
        // Creating a string via creating
        // object of StringBuffer class
        StringBuffer s = new StringBuffer("GeeksGeeks");

        // Invoking reverse() method
        s.reverse();

        // Returns "skeeGrofskeeG"
        System.out.println(s);
    }
}
```

Output

```
skeeGskeeG
```

Example 5: delete( ) and deleteCharAt()

It can delete characters within a StringBuffer by using the methods **delete( )** and **deleteCharAt( )**.The **delete( )** method deletes a sequence of characters from the invoking object. Here, the start Index specifies the index of the first character to remove, and the end Index specifies an index one past the last character to remove. Thus, the substring deleted runs from start Index to endIndex–1. The resulting StringBuffer object is returned. The **deleteCharAt( )** method deletes the character at the index specified by *loc.* It returns the resulting StringBuffer object.

Syntax:

```
StringBuffer delete(int startIndex, int endIndex)
StringBuffer deleteCharAt(int loc)
```

Java

```java
// Java Program to Illustrate StringBuffer class
// via delete() and deleteCharAt() Methods

// Importing I/O classes
import java.io.*;

// Main class
class GFG {

    // Main driver method
    public static void main(String[] args)
    {
        StringBuffer s = new StringBuffer("GeeksforGeeks");

        s.delete(0, 5);
        // Returns forGeeks
        System.out.println(s);

        s.deleteCharAt(7);
        // Returns forGeek
        System.out.println(s);
    }
}
```

Output

```
forGeeks
forGeek
```

Example 6: replace()

It can replace one set of characters with another set inside a StringBuffer object by calling replace( ). The substring being replaced is specified by the indexes start Index and endIndex. Thus, the substring at start Index through endIndex–1 is replaced. The replacement string is passed in str. The resulting StringBuffer object is returned.

Syntax:

```
StringBuffer replace(int startIndex, int endIndex, String str)
```

Example

Java

```
// Java Program to Illustrate StringBuffer class
// via replace() method

// Importing I/O classes
import java.io.*;

// Main class
class GFG {

    // Main driver method
    public static void main(String[] args)
    {
        StringBuffer s = new StringBuffer("GeeksforGeeks");
        s.replace(5, 8, "are");

        // Returns GeeksareGeeks
        System.out.println(s);
    }
}
```

Output

```
GeeksareGeeks
```

This article is contributed by **Lokesh Todwal.** Please write comments if you find anything incorrect, or if you want to share more information about the topic discussed above.

Last Updated : 03 Apr, 2023

Similar Reads

1. String vs StringBuilder vs StringBuffer in Java

2. Sorting collection of String and StringBuffer in Java
3. A Java Random and StringBuffer Puzzle
4. equals() on String and StringBuffer objects in Java
5. StringBuffer insert() in Java
6. StringBuffer deleteCharAt() Method in Java with Examples
7. StringBuffer appendCodePoint() Method in Java with Examples
8. StringBuffer delete() Method in Java with Examples
9. StringBuffer replace() Method in Java with Examples
10. StringBuffer reverse() Method in Java with Examples

Related Tutorials

1. Spring MVC Tutorial
2. Spring Tutorial
3. Spring Boot Tutorial
4. Java 8 Features - Complete Tutorial
5. Java IO Tutorial

Previous Next

Article Contributed By :

GeeksforGeeks

Vote for difficulty

Current difficulty : Easy

Easy | Normal | Medium | Hard | Expert

Improved By : ShreyasWaghmare, codingbeastsaysyadav, simmytarika5, anikakapoor, surinderdawra388, surindertarika1234, varshagumber28, kapoorsagar226, pujasingg43, paridhibhagwat2002, aditiyadav20102001, codearcade, faizghan7e8m, decode_aashish, rathoadavinash

Article Tags : Java-lang package, java-StringBuffer, Java-Strings, Java

Practice Tags : Java, Java-Strings

Report Issue

GeeksforGeeks
A-143, 9th Floor, Sovereign Corporate Tower, Sector-136, Noida, Uttar Pradesh - 201305

feedback@geeksforgeeks.org

Algorithms

Sorting

Searching

Greedy

Dynamic Programming

Pattern Searching

Recursion

Backtracking

Web Development

HTML

CSS

JavaScript

Bootstrap

ReactJS

AngularJS

NodeJS

Computer Science

GATE CS Notes

Operating Systems

Computer Network

Database Management System

Software Engineering

Digital Logic Design

Engineering Maths

Python

Python Programming Examples

Django Tutorial

Python Projects

Python Tkinter

OpenCV Python Tutorial

Python Interview Question

Data Science & ML

Data Science With Python

Data Science For Beginner

Machine Learning Tutorial

Maths For Machine Learning

Pandas Tutorial

NumPy Tutorial

NLP Tutorial

Deep Learning Tutorial

DevOps

Git

AWS

Docker

Kubernetes

Azure

GCP

Competitive Programming

Top DSA for CP

Top 50 Tree Problems

Top 50 Graph Problems

Top 50 Array Problems

Top 50 String Problems

Top 50 DP Problems

Top 15 Websites for CP

System Design

What is System Design

Monolithic and Distributed SD

Scalability in SD

Databases in SD

High Level Design or HLD

Low Level Design or LLD

Top SD Interview Questions

Interview Corner

Company Wise Preparation

GfG School

CBSE Notes for Class 8

Preparation for SDE

Experienced Interviews

Internship Interviews

Competitive Programming

Aptitude Preparation

CBSE Notes for Class 9

CBSE Notes for Class 10

CBSE Notes for Class 11

CBSE Notes for Class 12

English Grammar

Commerce

Accountancy

Business Studies

Economics

Management

Income Tax

Finance

UPSC

Polity Notes

Geography Notes

History Notes

Science and Technology Notes

Economics Notes

Important Topics in Ethics

UPSC Previous Year Papers

SSC/ BANKING

SSC CGL Syllabus

SBI PO Syllabus

SBI Clerk Syllabus

IBPS PO Syllabus

IBPS Clerk Syllabus

Aptitude Questions

SSC CGL Practice Papers

Write & Earn

Write an Article

Improve an Article

Pick Topics to Write

Write Interview Experience

Internships

Video Internship